

Planning for Optimal Feedback Control in the Volume of Free Space

Dmitry Yershov

Michael Otte

Emilio Frazzoli

Abstract—The problem of optimal feedback planning among obstacles in d -dimensional configuration spaces is considered. We present a sampling-based, asymptotically optimal feedback planning method. Our method combines an incremental construction of the Delaunay triangulation, volumetric collision-detection module, and a modified Fast Marching Method to compute a converging sequence of feedback functions. The convergence and asymptotic runtime are proven theoretically and investigated during numerical experiments, in which the proposed method is compared with the state-of-the-art asymptotically optimal path planners. The results show that our method is competitive with the previous algorithms. Unlike the shortest trajectory computed by many path planning algorithms, the resulting feedback functions can be used directly for robot navigation in our case. Finally, we present a straightforward extension of our method that handles dynamic environments where obstacles can appear, disappear, or move.

I. INTRODUCTION

We present the Asymptotically-optimal Control over Incremental Delaunay Simplicial Complexes (ACIDIC) method for computing an optimal feedback control among obstacles in d -dimensional configuration spaces. In this method, a sample sequence, the Delaunay triangulation, and a volumetric collision-detection module are used to build a simplicial free space approximation. This volumetric approximation parallels the Probabilistic RoadMap (PRM) [16] in that it captures the topology of the free space and has the same asymptotic computational complexity as its optimal implementation PRM* [15]. Contrary to the PRM and PRM*, using the Fast Marching Method (FMM) [28] on this simplicial approximation enables planning for a near-optimal path through the volume of d -dimensional cells instead of constraining it to edges of an 1D graph; see Fig. 1. Moreover, borrowing ideas from graph search literature [19], [20], [31], we extend the FMM to handle vertex insertions and deletions, which led us to efficient asymptotically optimal feedback planning and replanning algorithms.

For almost three decades, computing the optimal robot motion has been a central problem in robotics. Optimal solutions are required when resources such as fuel and time are expensive or limited. However, the optimal planning problem is computationally hard. For example, a relatively “easy” problem of finding the shortest path in a 3-dimensional polygonal environment is already PSPACE-hard [5]. Thus, only approximate optimal paths are computed in practice.

This work was supported by AFOSR FA8650-07-2-3744, ONR MURI N00014-09-1-1051, and ARO MURI W911NF-11-1-0046.

The Authors are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139, USA [yershov, ottemw, frazzoli]@mit.edu

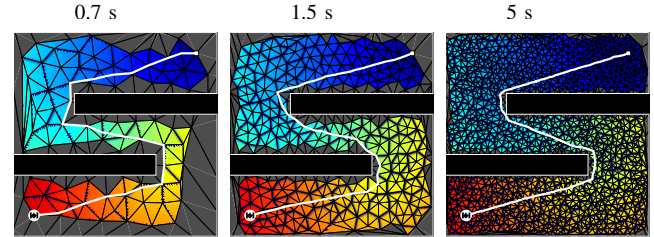


Fig. 1: Incremental feedback plan (color) and resulting path (white) created by the proposed method for a robot (lower left) that desires to reach a goal (upper right) while avoiding obstacles (black). Regions for which a policy is still unknown are gray. Movies, including replanning in dynamic environments, appear at <http://tinyurl.com/qjnazvr>

Recently, there has been a flurry of interest in sampling-based methods that provide *asymptotic optimality* by improving iteratively near-optimal solutions. For example, RRG* [15] constructs a rapidly-exploring random graph (RRG) using a sample sequence of random vertices. In practice, such algorithms can be terminated after a sufficiently accurate solution has been found. In this respect, they are similar to anytime graph-search algorithms [20]. Contrary to graph-search algorithms, the planners such as RRG converge to the true optimal path in the limit, as the iteration number tends to infinity.

Most sampling-based planning algorithms are *path-centric*, that is, they find an open-loop control that defines a near-optimal trajectory from start to goal [1], [10], [24], [25]. When used for robot navigation, such algorithms require an additional path-following controller to close the control loop. On the other hand, *control-centric* algorithms compute a feedback function that stabilizes the dynamical system towards the goal—eliminating the need of auxiliary controllers. Nonoptimal feedback planning algorithms are well established in robotics [21], [27], [32]. The first incremental, sampling-based, asymptotically optimal feedback stochastic control planner is presented in [13]. Introduced in [34], Simplicial Dijkstra and A* algorithms compute approximate optimal feedback control using a simplicial free space decomposition and the FMM, which recently have been combined with the adaptive mesh refinement yielding an asymptotically optimal, but not sampling-based feedback planning algorithm [33].

Our algorithm is different from the previous attempts to combine sampling strategies with the FMM. In [12], for example, the RRT algorithm is used to compute the initial path, which is improved during the post-processing using the FMM over a

local Delaunay triangulation. Since only a local approximation is used, the convergence to the globally optimal path is not guaranteed. Despite a naming similarity, the Fast Marching Trees (FMT*) algorithm [14] does not implement the FMM. The FMT* computes cost-to-go wavefront propagation using its values from the neighboring vertices instead of interpolating between them. Thus, it is closer in spirit to Dijkstra's graph search algorithm.

To the best of our knowledge, the ACIDIC is the first asymptotically optimal feedback planning method that combines sampling strategies, volumetric free space approximations, and efficient FMM-based feedback planning algorithms.

II. PRELIMINARIES

A. Optimal Feedback Planning Problem

An optimal control formulation provides the most natural framework for the feedback planning algorithm. Let X be the configuration space of a robot, that is, the complete set of parameters and their respective ranges that uniquely determine robot's internal configuration and position in the world. The open *obstacle set*, $X_{\text{obs}} \subset X$, corresponds to all states of the robot that result in either self-collisions or collisions with obstacles in the world. The robot is free to traverse the *free space*, $X_{\text{free}} = X \setminus X_{\text{obs}}$, along trajectories of an ordinary differential equation (ODE) with control

$$\dot{x}(t) = f(x(t), u(t)). \quad (1)$$

This equation defines the dynamics of the robot and allows the *input signal* $u(t)$, chosen from the *input set* U , to control it. Finally, we assume that X_{free} is a compact Lipschitz domain, that is, the boundary of X_{free} can be represented locally as a level-set of a Lipschitz continuous function.

In many problems in robotics the configuration space X and the motion model (1) are well-defined due to careful mechanism design. The obstacle set, on the other hand, inherits the complexity of the surrounding world, which is less predictable and may also evolve over time.

The performance of the robot is measured with respect to an additive cost functional:

$$J(\tilde{x}, \tilde{u}) = \int_0^{t_f} c(\tilde{x}(t), \tilde{u}(t)) dt \quad (2)$$

if $\tilde{x}(t) \in X_{\text{free}}$ for all $t \in [0, t_f]$, and $J(\tilde{x}, \tilde{u}) = \infty$ otherwise. In the above, \tilde{u} and \tilde{x} are realizations of the input signal and the corresponding trajectory, $c : X \times U \rightarrow \mathbb{R}^+$ is a positive local cost, and t_f is the terminal time. Usually, t_f is the first moment when $\tilde{x}(t_f)$ is inside the *goal set* X_{goal} . We assume, X_{goal} is a compact Lipschitz domain in X_{free} . The task of optimal motion planning algorithm is to minimize J .

We employ Bellman's dynamic programming principle and find the optimal control using the *cost-to-go* function V . At all points $x \in X$, $V(x)$ is defined as the optimal cost of reaching the goal from x , and it satisfies Hamilton-Jacobi-Bellman partial differential equation (HJB PDE):

$$\min_{u \in U} \{\nabla V(x) \cdot f(x, u) + c(x, u)\} = 0. \quad (3)$$

Once the cost-to-go is computed, the feedback control is given as a minimizing argument in (3), $\pi(x) = \arg \min_{u \in U} \{\nabla V(x) \cdot f(x, u) + c(x, u)\}$. The optimal trajectory is then found by replacing u with $\pi(x)$ in (1) and integrating the resulting ODE on the interval $[0, t_f]$.

B. HJB Numerical Discretization

When an analytical solution is unavailable for (3), we must resort to numerical PDE solvers. To this end, we follow closely the FMM discretization [28].

Definition 1 (Sample Set): A finite or countable set of distinct vertices \hat{X} is called a *sample set* of X .

Definition 2 (Abstract and Geometric Simplices): An *abstract simplex* of dimension d is a set of $(d + 1)$ vertices $\tau = \{x_0, \dots, x_d\} \subset \hat{X}$ such that $x_k \neq x_{k'}$ for all k and k' . A *geometric realization* of τ is a convex hull of its vertices, which we call *geometric simplex* and denote X_τ . Formally,

$$X_\tau = \left\{ \sum_{k=0}^d \alpha_k x_k \mid \forall \alpha_k \geq 0 \text{ such that } \sum_{k=0}^d \alpha_k = 1 \right\}. \quad (4)$$

In the above, $\{\alpha_k\}_{k=0}^d$ are called *barycentric coordinates* of a point $x = \sum_{k=0}^d \alpha_k x_k \in X_\tau$. Finally, $X_{\tau'}$ is called a (*proper*) *facet* of X_τ if τ' is a (*proper*) subset of τ .

Definition 3 (Vertex Set Triangulation): A triangulation of \hat{X} is a set of abstract d -dimensional simplices \mathcal{T} such that for all $\tau, \tau' \in \mathcal{T}$ and $\tau \neq \tau'$ the intersection $X_\tau \cap X_{\tau'}$ is a proper facet of both simplices and the union $\bigcup_{\tau \in \mathcal{T}} X_\tau$ covers the convex hull of \hat{X} .

The numerical solution is computed in three steps.

First, we consider a triangulation of a sample set of vertices in X . An approximation of X_{free} is then derived from this vertex set triangulation by ignoring simplices that have a nonempty intersection with X_{obs} . Note that this approximation approaches X_{free} in the limit, as the dispersion of the vertex set tends to zero.

Second, using a collision-free triangulation of X_{free} , we define a piecewise linear cost-to-go approximation:

$$\hat{V}(x) = \sum_{x_k \in \tau} \hat{V}(x_k) \alpha_k(x), \quad (5)$$

in which τ is an abstract simplex such that $x \in X_\tau$, and $\alpha_k(x)$ is k th barycentric coordinate of x in X_τ . Note that \hat{V} is defined completely using its values at sampled vertices.

Finally, we substitute (5) into (3) and derive the discrete Hamilton-Jacobi-Bellman equation at vertex $x \in \hat{X}$:

$$\min_{\tau \in \text{St}(x)} \min_{u \in U_{x,\tau}} \{\nabla_\tau \hat{V}(x) \cdot f(x, u) + c(x, u)\} = 0. \quad (6)$$

In the above, $\nabla_\tau \hat{V}$ is a restriction of $\nabla \hat{V}$ onto simplex X_τ and $\text{St}(x) = \{\tau \mid x \in \tau\}$ is called a *star* of vertex x and represents a local neighborhood around x . The minimization over input signal u is constrained to the set $U_{x,\tau} =$

$\{u \in U \mid \exists \delta > 0 : x + \delta f(x, u) \in X_\tau\}$. This constraint is necessary to construct a *positive coefficient* discretization, which converges to the viscosity solution of (3) [9].

The linear convergence rate of the numerical discretization (6) has been established in [34], that is, $E \leq Ch$, in which $E = \sup_{x \in X} |\hat{V}(x) - V(x)|$ is the global numerical error, and $h = \sup_{\tau \in \mathcal{T}} \max_{x, x' \in \tau} \|x - x'\|$ is the maximum edge length. In the above, $C > 0$ is a constant, which is independent of h .

When considered at all vertices $x \in \hat{X}$, equation (6) defines a system of nonlinear equations. Similar to Dijkstra's algorithm, the Fast Marching Methods (FMM) [28] evaluates the cost-to-go values in a single pass through the vertex set using a priority queue. If the triangulation is acute, then the FMM solves this nonlinear system [18], [29].

C. Delaunay Triangulation

Definition 4 (Delaunay Triangulation): The Delaunay (also known as Delone) triangulation, \mathcal{T} , of a sample set \hat{X} is such that, for all $\tau \in \mathcal{T}$ and all $x \in \hat{X} \setminus \tau$, x is located outside of the circumsphere of X_τ .

General vertex set triangulations are difficult to compute, with the notable exception of the Delaunay triangulation (DT). In computational geometry, the bijection between Delaunay simplices in \mathbb{R}^d and lower faces of the convex hull of sample vertices lifted onto a paraboloid in \mathbb{R}^{d+1} has been established. Thus, geometric convex hull algorithms can be used for constructing the DT. It also follows from this relation that the DT exists and is unique if sample vertices are in *general position* (that is, there are no $d+2$ vertices such that they belong to some d -dimensional sphere).

Moreover, compared with other vertex set triangulations, the DT maximizes the min-containment radius of simplices [26] and minimizes the second-order error of a piecewise linear interpolation in 2D [3] and in d -dimensional case [6]. The former implies that the DT is the most regular of all triangulations, and the latter guarantees the smallest possible numerical error in (6).

Considering the simplicity of computing the DT and its optimality with respect to interpolation error, we use the DT for approximating the cost-to-go function on X_{free} .

D. Stochastic Delaunay Triangulation

Random sampling has been proven useful for high-dimensional path planning problems. Randomized motion planning algorithms have both theoretical and practical advantages. In theory, they are probabilistically complete, which guarantees finding an existent solution. In practice, they rapidly explore the environment searching for a solution.

Following this trend, we present the Delaunay triangulation of a random vertex set, which we call *stochastic Delaunay triangulation* (SDT). From the motion planning perspective, the SDT can be compared with the PRM in that they both approximate the free space, as the number of sample vertices increases. Unlike the PRM graph, however, the SDT is a volumetric approximation of X_{free} . Although computing simplex collisions is generally more difficult than performing an 1-dimensional collision check, this

volumetric information enables finding paths that traverse through simplicial cells instead of being constrained on graph edges.

In this section, we establish a connection between the SDT and Poisson-Delaunay mosaics and some useful properties of the former that follow from the integral geometry.

Definition 5 (Poisson Point Process in \mathbb{R}^d): A Poisson point process in \mathbb{R}^d of intensity ρ is a set of random points such that for two bounded, measurable, disjoint sets X_1 and X_2 , the number of points of this process that are inside X_1 and X_2 are two independent Poisson random variables with parameter $\rho\mu(X_1)$ and $\rho\mu(X_2)$, respectively.¹

Definition 6 (Poisson-Delaunay Mosaic): Let \hat{X} be a realization of a Poisson process. The Delaunay triangulation of \hat{X} is a Poisson-Delaunay mosaic.²

It follows from Definition 5 that a set of N uniformly and independently distributed vertices from the free space is the restriction of the Poisson point process. Using maximum likelihood estimate, we establish the intensity of this process: $\rho = N/\mu(X_{\text{free}})$. This relation shows that the SDT can be considered as the restriction of a Poisson-Delaunay mosaic.

In integral geometry, statistical properties of Poisson-Delaunay mosaics have been established. For example, the expected number of simplices in the star of a vertex depends on the dimension number d , but it is independent of the intensity ρ [17]. Therefore, the expected number of vertex-adjacent simplices is independent of N for the SDT.³

Another property of Poisson-Delaunay mosaics is related to the average simplex size as a function of the point process intensity. The average edge length is $\Theta(\rho^{-1/d})$ [23], and the expected maximum edge length is $\Theta((\log \rho/\rho)^{1/d})$ [4]. Using previously derived relation between ρ and N , we find that the expected edge length and its maximum of the SDT are $\Theta(N^{-1/d})$ and $\Theta((\log N/N)^{1/d})$, respectively.⁴

III. SAMPLING-BASED FEEDBACK PLANNING ALGORITHM

We now present the ACIDIC method for sampling-based feedback planning. The execution trace of our method is similar to most sampling-based path-centric planners:

- 1) Sample a new vertex x_{new} from X ;
- 2) Refine the Delaunay triangulation to include x_{new} ;
- 3) Update the cost-to-go values and associated feedback control in the simplicial approximation of X_{free} .

At a conceptual level, the ACIDIC method “etches” the free space away from the obstacle set while simultaneously refining a feedback control. In the limit of infinitely many sampled vertices, all points of X_{free} become part of the triangulation and the optimal feedback control is computed.

¹Here, $\mu(X)$ denotes the Lebesgue measure of X .

²The definition of DT can be extended to locally sparse infinite sets.

³Note that our optimal planner has a constant branching factor in expectation compared with $O(\log N)$ branching factor for RRT*.

⁴Note that exactly the same bound on the shrinking radius enables percolation limit in the RRG-based algorithms.

Algorithm 1 ACIDIC method (planning version)

Input: Compact Lipschitz domains $X_{\text{goal}} \subseteq X_{\text{free}} \subseteq X$;
bounding simplex τ_0 such that $X_{\text{free}} \subset X_{\tau_0}$
Output: Yields $\hat{\pi}$
1: **global** \hat{V} , DT, τ_0 , QP
2: Let $\mathcal{C}(\tau_0) \leftarrow \emptyset$, $\mathcal{T} \leftarrow \{\tau_0\}$, and DT $\leftarrow (\mathcal{T}, \mathcal{C}, \text{St})$
3: Initialize $\hat{V}(x) \leftarrow \infty$ for all $x \in \tau_0$
4: **loop**
5: $x_{\text{new}} \leftarrow \text{SamplePoint}()$
6: **InsertPoint**(x_{new})
7: **InitiateFMM**($\{x_{\text{new}}\}$)
8: **CompleteFMM**()
9: **yield** $\hat{\pi}(x) = \arg \min_{u \in U} \{\nabla \hat{V}(x) \cdot f(x, u) + c(x, u)\}$

Algorithm 2 ACIDIC method (replanning version)

Input: Compact Lipschitz domains $X_{\text{goal}} \subseteq X_{\text{free}} \subseteq X$;
bounding simplex τ_0 such that $X_{\text{free}} \subset X_{\tau_0}$
Output: Yields $\hat{\pi}$
1: **global** \hat{V} , DT, τ_0 , QP
2: Let $\mathcal{C}(\tau_0) \leftarrow \emptyset$, $\mathcal{T} \leftarrow \{\tau_0\}$, and DT $\leftarrow (\mathcal{T}, \mathcal{C}, \text{St})$
3: Initialize $\hat{V}(x) \leftarrow \infty$ for all $x \in \tau_0$
4: **loop**
5: **if** Moving Robot **then**
6: Move Robot
7: $x_{\text{new}} \leftarrow \text{SamplePoint}()$; $X_{\text{set}} \leftarrow \{x_{\text{new}}\}$
8: **InsertPoint**(x_{new})
9: **if** Obstacle Changes Observed **then**
10: $X_{\text{set}} \leftarrow X_{\text{set}} \cup (\bigcup \{\tau \mid X_{\tau} \cap \Delta X_{\text{obs}} \neq \emptyset\})$
11: **InitiateFMM**(X_{set})
12: **CompleteFMM**()
13: **yield** $\hat{\pi}(x) = \arg \min_{u \in U} \{\nabla \hat{V}(x) \cdot f(x, u) + c(x, u)\}$

* Note that we disallow cost-to-go values at two different vertices to depend on each other. Thus, in **VertexLookAhead**(), it is assumed $\hat{V}(x') = \infty$ at all $x' \in \tau$ such that $\hat{V}(x') = \text{minloc}(x', \tau, \hat{V})$ when computing $\hat{V}^*(x)$.

Algorithm 3 **InsertPoint**(x)

1: $\mathcal{T}_{\text{vis}} \leftarrow \text{ComputeVisible}(x)$
2: $\mathcal{R} \leftarrow \text{ComputeRidges}(\mathcal{T}_{\text{vis}})$
3: **for all** $\rho \in \mathcal{R}$ **do**
4: $\tau \leftarrow \rho \cup \{x\}$; $\mathcal{T} \leftarrow \mathcal{T} \cup \{\tau\}$; $\mathcal{C}(\tau) \leftarrow \emptyset$
5: **for all** $\tau' \in \mathcal{T}_{\text{vis}}$ **do**
6: $\mathcal{C}(\tau') \leftarrow \mathcal{C}(\tau') \cup \{\tau\}$
7: **for all** $x' \in \rho$ **do**
8: $\text{St}(x') \leftarrow (\text{St}(x') \cup \{\tau\}) \setminus \mathcal{T}_{\text{vis}}$
9: $\text{St}(x) \leftarrow \text{St}(x) \cup \{\tau\}$

Algorithm 4 **ComputeVisible**(x)

1: Initialize LIFO queue Q and $\mathcal{T}_{\text{vis}} \leftarrow \emptyset$
2: $\tau \leftarrow \text{FindSimplex}(x)$
3: **Push**(Q, τ)
4: **while** $Q \neq \emptyset$ **do**
5: **Pop**(Q, τ)
6: **if** **IsVisible**(x, τ) **then**
7: $\mathcal{T}_{\text{vis}} \leftarrow \mathcal{T}_{\text{vis}} \cup \{\tau\}$
8: **for all** $\tau' \in \bigcup \{\text{St}(x)\}_{x \in \tau} \setminus \mathcal{T}_{\text{vis}}$ **do**
9: **Push**(Q, τ')
10: **return** \mathcal{T}_{vis}

Algorithm 5 **ComputeRidges**(\mathcal{T}_{vis})

1: $\mathcal{R} \leftarrow \emptyset$
2: **for all** $\tau \in \mathcal{T}_{\text{vis}}$ **do**
3: **for all** $x \in \tau$ **do**
4: $\rho \leftarrow \tau \setminus \{x\}$
5: **if** $\nexists \tau' \in \mathcal{T}_{\text{vis}} \setminus \{\tau\} : \rho \subset \tau'$ **then**
6: $\mathcal{R} \leftarrow \mathcal{R} \cup \{\rho\}$
7: **return** \mathcal{R}

Algorithm 6 **FindSimplex**(x)

1: $\tau \leftarrow \tau_0$
2: **while** $\mathcal{C}(\tau) \neq \emptyset$ **do**
3: **for** $\tau' \in \mathcal{C}(\tau)$ **do**
4: **if** $x \in X_{\tau'}$ **then**
5: $\tau \leftarrow \tau'$
6: **break for loop**
7: **return** τ

Algorithm 7 **InitiateFMM**(X_{set})

1: **for all** $x \in X_{\text{set}}$ **do**
2: $\hat{V}(x) \leftarrow \infty$
3: **Remove**(QP, x)
4: **for all** $x \in X_{\text{set}}$ **do**
5: **VertexLookAhead**(x)
6: **UpdateQueue**(QP, x)

Algorithm 8 **VertexLookAhead**(x)*

1: $\hat{V}^*(x) \leftarrow \infty$
2: **for** $\tau \in \text{St}(x)$ **do**
3: **if** **IsCollisionFree**(τ) **then**
4: $\hat{V}^*(x) \leftarrow \min\{\text{minloc}(x, \tau, \hat{V}), \hat{V}^*(x)\}$

Algorithm 9 **CompleteFMM**()

1: **while** QP $\neq \emptyset$ **do**
2: $x \leftarrow \text{Pop}(\text{QP})$
3: **if** $\hat{V}(x) > \hat{V}^*(x)$ **then**
4: $\hat{V}(x) \leftarrow \hat{V}^*(x)$
5: **else if** $\hat{V}(x) < \hat{V}^*(x)$ **then**
6: $\hat{V}(x) \leftarrow \infty$
7: **for all** $x' \in \bigcup \{\tau\}_{\tau \in \text{St}(x)} \setminus \{x\}$ **do**
8: **VertexLookAhead**(x')
9: **UpdateQueue**(QP, x')
10: **if** $\hat{V}(x) = \infty$ **then**
11: **VertexLookAhead**(x)
12: **UpdateQueue**(QP, x)

Algorithm 10 **UpdateQueue**(QP, x)

1: **if** $\hat{V}^*(x) \leq \hat{V}(x) - \epsilon$ **or** $\hat{V}^*(x) > \hat{V}(x)$ **then**
2: **if** $x \notin \text{QP}$ **then**
3: **Push**(QP, x)
4: $K(x) \leftarrow \min\{\hat{V}(x), \hat{V}^*(x)\}$
5: **else if** $x \in \text{QP}$ **then**
6: **Remove**(QP, x)

This basic idea can be used to create an entire family of feedback planning algorithms. The behavior of a particular algorithm depends on three factors: the sampling strategy, the volumetric decomposition (that also includes collision-detection strategy), and the methods by which the approximate cost-to-go function and feedback control are updated.

We discuss our implementation of these three steps in Sections III-A–III-C, respectively. An illustrative example is presented in Section III-C.1, and an extension to replanning in dynamic environments in Section III-C.2. Our generic implementation can be improved for specific applications.

A. Sampling Strategy

Motivated by theoretical results from Section II, we explored three different sampling strategies, which we implemented in Algorithm 1: uniform random sampling, deterministic largest-

simplex Delaunay refinement, and goal-oriented refinement in the vicinity of the current shortest path and near obstacle boundaries.

In theory, uniform random sampling guarantees regularity and convergence of the DT in expectation. In practice, however, biased or carefully engineered deterministic sequences are likely to improve planning algorithm performance by constructing high-quality triangulations.

The largest circumradius of all Delaunay simplices defines the Euclidean dispersion of the sampled vertex set. Moreover, the circumcenter of the corresponding simplex maximizes the distance to the closest sample vertex. By placing x_{new} at this circumcenter, we optimize the sample sequence to consider unexplored regions of the configuration space. This strategy is known as Delaunay refinement [7], and is proven to produce a high-quality triangulation [30].

In the limit, Delaunay refinement samples vertices uniformly

in the configurations space. However, the convergence of our algorithm can be improved by sampling regions in which the interpolation error is the highest. We consider, for example, regions where the cost function is highly nonlinear, such as near obstacle boundaries or the goal set as well as the vicinity of the current optimal path. In particular, we choose x_{new} as the circumcenter of the largest circumsphere of Delaunay simplices that are either on the boundary of X_{obs} and X_{goal} or contain the current shortest path.

B. Computing Volumetric Free Space Approximation

At every iteration, Algorithm 3 uses a newly sampled vertex to improve the volumetric approximation of X_{free} . To update the Delaunay triangulation, we follow closely the incremental convex hull algorithm introduced in [2]. To this end, we find all simplices that are *visible* from the newly inserted vertex; see Algorithm 4. Next, we find a set of *ridges*, which are defined as faces separating visible and invisible simplices; see Algorithm 5. A set of new Delaunay simplices is created by connecting all ridges with the inserted vertex; see Line 4 in Algorithm 3. Finally, we “remove” visible simplices by updating their children sets to include all newly inserted simplices.

Note that all simplices are actually retained; however, only childless simplices belong to the current DT, which is reflected in updating the local connectivity information in $\text{St}(x)$ and $\text{St}(x')$; see Lines 7–9 in Algorithm 3. The remaining simplices organized into a directed acyclic graph structure that helps locating future vertices within the Delaunay triangulation, as it is prescribed by Algorithm 6.

After the DT is updated, a black box collision-detection module is used to find free-to-traverse simplices in the current triangulation. We assume the conservative collision-detection implementation, that is, the simplex is considered collision-free if $X_{\tau} \cap X_{\text{obs}} = \emptyset$.

It is worth noting that pointwise collision-detection modules are ill-suited for volumetric representations because each simplex contains an infinite number of points to check. Thus, as it is also the case with many graph-based planning algorithms, we require additional information about the obstacle set to compute the volumetric free space approximation. For example, if the distance to the nearest obstacle is known and higher-order dynamics, such as velocities, are bounded, then we can verify collision-free simplices by solving a convex minimization problem. Volumetric collision-detection is an advanced topic, and further implementation details are beyond the scope of the current paper.

C. Computing Cost-To-Go Function and Feedback Control

The cost-to-go function and the optimal feedback control are maintained using a modified version of the Fast Marching Method (FMM). In particular, our version addresses a nonmonotonic cost-to-go wavefront propagation, which may be caused by either a nonacute triangulation or a part of the free space approximation becoming the obstacle set due to local simplex rewiring and a conservative collision-detection.

Our modifications to the FMM follow closely replanning path-centric strategies, such as D* Lite [19] and RRT^x [24], which deal with appearing and disappearing obstacles by propagating the *increase* wavefront ahead of the *decrease* wavefront. To prevent infinite loops, the algorithm interrupts wavefront propagation when changes are smaller than a given parameter ϵ . At the end of this section, we investigate a peculiar side effect of our implementation: a straightforward extension to an optimal feedback replanning algorithm.

1) *Fast Marching Feedback Planning Algorithm*: The ACIDIC planning algorithm (Algorithm 1) improves incrementally the feedback control by initiating and propagating the cost-to-go wavefront (Lines 7 and 8, respectively).

The wavefront is initiated at newly sampled vertex x_{new} , when the look-ahead value \hat{V}^* becomes inconsistent with the current cost-to-go value \hat{V} ; see Algorithm 7. The look-ahead value is computed in Algorithm 8 using `minloc` function that solves the inner minimization problem in (6) for collision-free simplices. The implementation of `minloc` and its geometric interpretation are discussed in [34]. Finally, we also assume `minloc` returns 0 if x is in the goal set.

The inconsistency between \hat{V} and \hat{V}^* implies that the solution of the nonlinear system is not found yet. Algorithm 9 repairs inconsistency by propagating the cost-to-go wavefront through the environment in the fast marching fashion. To this end, the vertices are organized in a priority queue in the increasing order of their key values $K = \min\{\hat{V}, \hat{V}^*\}$. For each vertex x , two cases are considered: 1) cost-to-go value decreases (Line 3) and 2) cost-to-go value increases (Line 5). In the first case, $\hat{V}(x)$ is updated to its current best estimate, $\hat{V}^*(x)$, and the decrease wavefront is propagated to vertex neighbors lowering their \hat{V} values. In the second case, $\hat{V}(x)$ is set temporarily to infinity causing \hat{V}^* values to increase at all neighbors whose cost-to-go values are depended on $\hat{V}(x)$. Thus, the increase wavefront is propagated. Next, $\hat{V}^*(x)$ is updated creating the decrease wavefront, which repairs inconsistencies introduced by the increase wavefront.

When Algorithm 9 terminates, the cost-to-go function is consistent with (6) up to a small ϵ error in the entire domain. However, if the initial position is known the global cost-to-go computations are unnecessary, and the estimate of the cost-to-come can be used to restrict the computation domain. Various heuristics and their effect on computations are discussed in [8], [34].

2) *Fast Marching Replanning Algorithm*: The goal of the ACIDIC replanning algorithm is to update the feedback control as soon as robot’s sensors detect a change in obstacle configuration. Fast control loop relies crucially on replanning algorithm efficiency. In this case obstacles may appear or disappear, which results in increasing or decreasing cost-to-go values. Fortunately, ACIDIC planning algorithm accounts for both of these changes, and its extension towards replanning algorithm is rather straightforward.

We present Algorithm 2, in which wavefront propagation is

initiated once the change in obstacle set is confirmed (Line 10). A volumetric collision-detection module is used to find all inconsistent vertices that are affected by obstacle changes. This conservative estimate guarantees that after wavefront propagation the computed cost-to-go function is consistent with (6) up to a small ϵ error.

IV. ANALYSIS OF THE ALGORITHM

A. Numerical Convergence

The resolution of the sample set increases when each additional vertex is inserted. Hence, the accuracy of the planning algorithm is expected to improve. This intuition can be rigorously supported combining the results of Section II.

Theorem 7 (Numerical Convergence): We assume $\epsilon = 0$ and N random vertices are sampled. The expected solution error, $E[\mathcal{E}]$, is then $O((\frac{\log N}{N})^{\frac{1}{d}})$.

Proof: The proof follows from the numerical error bound presented in Section II-B and the expected maximum edge length presented in Section II-D. ■

B. Computational Complexity

To establish the computational complexity of Algorithm 1, we consider its expected runtime per iteration.

It should be noted that the worst-case runtime of our algorithm is bounded from below by the maximum number of Delaunay simplices in the triangulation of N vertices, which is proportional to $N^{\lceil d/2 \rceil}$ [22]. However, such artificially constructed cases “rarely” occur in practice. Thus, algorithms that we proposed for vertex insertion and wavefront propagation are *output-sensitive* in that they have optimal complexity $O(\log N)$ per Delaunay simplex. In conjunction with the results from Section II-D we prove the expected runtime bounds for the ACIDIC method.

Lemma 8: Let \mathcal{T} be a DT of a random vertex set \hat{X} , and let $x_{\text{new}} \in X$. For the closest vertex $x^* = \arg \min_{x \in \hat{X}} \|x_{\text{new}} - x\|$, there exists $\tau \in \text{St}(x^*)$ such that x is inside the circumsphere of X_τ .

Proof: We omit the proof due to space limitations. ■

Theorem 9 (Delaunay Triangulation Complexity): The expected runtime of Algorithm 3 is $O(\log N)$ per iteration.

Proof: Since the expected size of $\text{St}(x)$ is independent of N , the visible simplex number and the ridge number are constant in expectation. Thus, average simplex rewiring runtime is constant. Moreover, from Lemma 8, it follows that finding the first visible simplex can be done in expected $O(\log N)$ time using Kd-trees [11]. ■

Note that our algorithm for finding the first visible simplex avoids using Kd-trees for simplicity. In theory, it is not yet clear that the average depth of the children graph is $O(\log N)$. However, this algorithm performs well in practice.

Theorem 10 (Wavefront Propagation Complexity): The expected runtime of Algorithm 9 is $O(\log N)$ for all $\epsilon > 0$.

Proof: Since the approximate cost-to-go function converges, the number of ϵ -changes of \hat{V} at each vertex is bounded. Thus, the number of times wavefront propagates through each

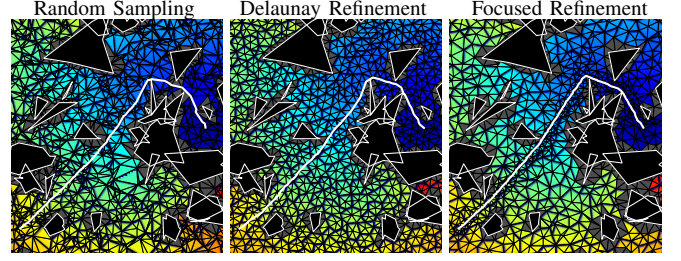


Fig. 2: Temperature map of the cost-to-go values (color) and path (white) computed by ACIDIC algorithm during the first 10 seconds in a randomly generated 2D environment of size 100×100 . Delaunay simplices are outlined in black, and obstacles are solid black.

vertex is constant. Maintaining the priority queue, however, takes $O(\log N)$ time. ■

V. EXPERIMENTS, RESULTS, AND DISCUSSION

The basic idea presented in this paper can be used to create an entire family of incremental volumetric feedback planning algorithms. For brevity, we limit our investigation to the implementations proposed in Section III. In Sections V-A and V-B we experimentally evaluate the performance of feedback planning in static environments and compare our results with state-of-the-art graph-based methods. In Section V-C we present simulations of the replanning version of the algorithm in a dynamic environment.

All experiments were run in simulated environments on a Dell Optiplex 790 with Intel i7 chip and 16GB of RAM; a single processor core is used for all experiments. All algorithms are implemented in Julia programming language. Both volumetric and graph-based algorithms use the same code-base in order to make the comparison as fair as possible. For example, all algorithms use the same sampling schemes, obstacle representations, collision-detection routines, heap data-structures, control-loop computations, data-logging procedures, and so on.

A. Point Robot in Random 2D Static Environment

In this experiment, a holonomic single integrator point-robot desires to travel to a goal point among randomly generated polygonal obstacles in 2D; see Fig. 2. We compare our ACIDIC planning algorithm with RRT* and RRT#. Results of most basic versions appear in Fig. 3, while a comparison of variants that use different sampling techniques and values of ϵ appears in Fig. 4.

In theory, volumetric and graph-based methods have identical asymptotic performance per iteration. In experiments, however, we observe that volumetric methods have a constant factor overhead associated with updating the Delaunay triangulation and computing fast marching wavefront propagation. Thus, our method processes fewer vertices per unit of wall time. Despite the difference in sampling performance, convergence rates of all considered methods are comparable, which implies that the ACIDIC method is more efficient in using sampled vertices. We

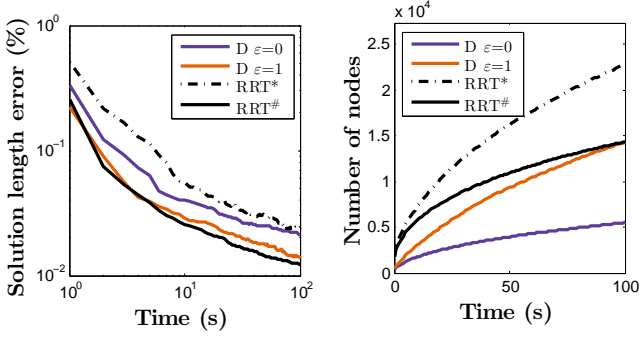


Fig. 3: Averaged over 50 trials relative path error (left, log-log scale) and the vertex number (right, regular scale) against wall time for ACIDIC algorithm using Delaunay refinement with $\epsilon = 0$ (purple) and $\epsilon = 1$ (orange), RRT* (black dot-dash), and RRT# (black solid) in the environment of Fig. 2.

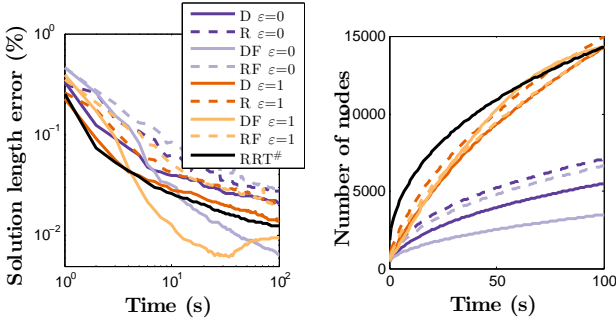


Fig. 4: Averaged over 50 trials relative path error (left, log-log scale) and the vertex number (right, regular scale) against wall time in the environment of Fig. 2. Results are for ACIDIC algorithm using Delaunay refinement (solid) and random sampling (dashed), $\epsilon = 0$ (dark/light purple) and $\epsilon = 1$ (orange/yellow), uniform (dark purple/orange) and focused (light purple/yellow) sampling, and RRT# (black).

attribute this efficiency to allowing the path traverse through the volume of space potentially ignoring all sampled vertices.

We optimized the convergence rate of the ACIDIC method using only the information gained from the volumetric decomposition. First, refining near obstacle boundaries is accomplished by sampling simplices that are partially in collision. This focused sampling enables the discovery of narrow corridors in the free space. Hence, the optimal solution can be found using fewer vertices. Second, the convergence rate is improved by refining the triangulation in the vicinity of the best path. Finally, truncating wavefront propagation reduces the runtime per iteration. The optimized ACIDIC method outperforms all graph-based algorithms considered in our experiments.

Note that truncating the wavefront propagation by setting $\epsilon = 1$ significantly improved the convergence in static environments. In graph-based methods, the usefulness of such this idea appears to be limited to dynamic environments, in which fast wavefront

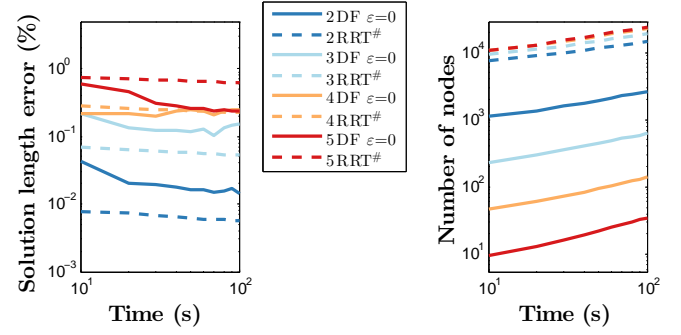


Fig. 5: Averaged over 50 trials relative path error (left, log-log scale) and the vertex number (right, log-log scale) against wall time for ACIDIC algorithm (solid) and RRT# (dashed) in 2D to 5D environments (color).

propagation and the control-loop speed are primary concerns. We believe the reason why the truncation benefits the convergence of ACIDIC methods is due to the larger constant factor associated with recomputing the cost-to-go function through the triangulation.

B. Block Obstacle in \mathbb{R}^d

In Fig. 5, we evaluate the performance of ACIDIC method in dimensions between 2 and 5. The environment used for each dimension is a hypercube with a single prismatic obstacle located at the center. The obstacle spans one-half of the environment in the first two dimensions and has infinite length along other dimensions.

As with many planning methods, the volumetric idea suffers from the “curse of dimensionality”. From the experiments, it became evident that the runtime per vertex grows exponentially with the dimension number for the ACIDIC algorithm. Note that this overhead is constant in any particular dimension, and the expected vertex insertion complexity for ACIDIC algorithm remains $O(\log N)$, in which N is the number of already inserted vertices. Opposite to the volumetric methods, the runtime per vertex is largely independent of the dimension number for graph-based algorithms, which, unfortunately, does not yield their faster convergence. From the convergence results, we confirm that volumetric methods use vertices more efficiently. We attribute this to the fact that $d+1$ vertices are sufficient to construct a simplex covering large volume of the free space. On the other hand, graph-based methods may require many more sample vertices to recover the shortest path.

C. Replanning Simulations in Dynamic Environments

Due to space limitations and also the fact that navigation through dynamic environment is best visualized in media that incorporates time, we have uploaded a playlist of movies to <http://tinyurl.com/qjnazvr> that illustrate the dynamic version of ACIDIC replanning algorithm.

The replanning version of ACIDIC is the first asymptotically-optimal sampling-based feedback replanner that is capable of repairing the feedback control in real-time when obstacles unexpectedly appear, disappear, or move within the configuration space. While RRT^X is closely related to our algorithm, it is path-centric and requires a feedback control to be computed in post-processing. While this can be done quickly, in practice, the particular method that is used will significantly affect reaction time — the most important performance measure of a replanning algorithm. Thus, we do not compare the two methods directly.

VI. SUMMARY

Proposed in this paper, is the first asymptotically optimal feedback planning algorithm that uses a volumetric approximation of free space in conjunction with the Fast Marching Method. The main idea of the method is to build an incremental Delaunay triangulation use a sample sequence and compute a feedback control in all collision-free simplices. The idea can be used for both planning in static environments and real-time replanning in dynamic environments.

We have established theoretical convergence guarantees and asymptotic per iteration computational complexity. During the experiments in simulated environments, we confirmed theoretical results and compared the performance of our implementation with that of state-of-the-art asymptotically optimal planners. It was established that the performance of a our basic implementation is similar to that of the previous planning algorithms. However, optimizing sampling and wavefront propagation routines proved beneficial for a substantial performance increase.

While being closely related to the previous asymptotically-optimal graph-based planners, such as, RRT^* , $\text{RRT}^\#$, and RRT^X , the ACIDIC method is fundamentally different from all of them. In particular, instead of computing an one-dimensional path on a graph, our method computes a collision-free feedback control that stabilizes the system towards the goal in the entire volume of the free space. Thus the output of the ACIDIC method can be used directly to control the system, and the implementation of auxiliary path-following controllers can be avoided. Analysis and experiments show that the ACIDIC method is theoretically sound and works well in practice.

ACKNOWLEDGMENT

We would like to thank Prof. Yuliy Baryshnikov from University of Illinois at Urbana-Champaign for introducing us to integral geometry, which enabled asymptotic complexity analysis of the proposed algorithm.

REFERENCES

- [1] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, May 2013, pp. 2421–2428.
- [2] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The Quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.*, vol. 22, no. 4, pp. 469–483, Dec. 1996.
- [3] M. Bern and D. Eppstein, "Mesh generation and optimal triangulation," *Computing in Euclidean geometry*, vol. 1, pp. 23–90, 1992.
- [4] M. Bern, D. Eppstein, and F. Yao, "The expected extremes in a Delaunay triangulation," *Int. J. Comput. Geom. Appl.*, vol. 01, no. 01, pp. 79–91, Mar. 1991.
- [5] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *Proceedings of 28th Annual Symposium on Foundations of Computer Science*, Oct. 1987, pp. 49–60.
- [6] L. Chen and J. Xu, "Optimal Delaunay triangulations," *Journal of Computational Mathematics*, vol. 22(2), pp. 299–308, 2004.
- [7] L. P. Chew, "Guaranteed-quality triangular meshes," Computer Science, Cornell University, Tech. Rep., Apr. 1989.
- [8] Z. Clawson, A. Chacon, and A. Vladimirovsky, "Causal domain restriction for Eikonal equations," Sep. 2013.
- [9] M. G. Crandall and P. L. Lions, "Viscosity solutions of Hamilton-Jacobi equations," *Transactions of the American Mathematical Society*, vol. 277, no. 1, pp. 1–42, May 1983.
- [10] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 18–47, Jan. 2014.
- [11] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, Sep. 1977.
- [12] J. V. Gómez, D. Álvarez, S. Garrido, and L. Moreno, "Improving sampling-based path planning methods with fast marching," in *ROBOT2013: First Iberian Robotics Conference*, ser. Advances in Intelligent Systems and Computing, M. A. Armada, A. Sanfeliu, and M. Ferre, Eds. Springer International Publishing, 2014, vol. 253, pp. 233–246.
- [13] V. A. Huynh, S. Karaman, and E. Frazzoli, "An incremental sampling-based algorithm for stochastic optimal control," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, May 2012, pp. 2865–2872.
- [14] L. Janson and M. Pavone, "Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions – extended version," Jul. 2013.
- [15] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, Jun. 2011.
- [16] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, Aug. 1996.
- [17] D. G. Kendall, "Random Delaunay simplexes in \mathbb{R}^m ," *Journal of Statistical Planning and Inference*, vol. 25, no. 3, pp. 225–234, Jul. 1990.
- [18] R. Kimmel and J. A. Sethian, "Computing geodesic paths on manifolds," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 95, no. 15, pp. 8431–8435, Jul. 1998.
- [19] S. Koenig and M. Likhachev, "D* Lite," in *Eighteenth National Conference on Artificial Intelligence*. Menlo Park, CA, USA: American Association for Artificial Intelligence, 2002, pp. 476–483.
- [20] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, May 2004.
- [21] S. R. Lindemann and S. M. LaValle, "Simple and efficient algorithms for computing smooth, collision-free feedback laws over given cell decompositions," *The International Journal of Robotics Research*, vol. 28, no. 5, pp. 600–621, May 2009.
- [22] P. McMullen, "The maximum numbers of faces of a convex polytope," *Mathematika*, vol. 17, pp. 179–184, Dec. 1970.
- [23] R. E. Miles, "A synopsis of 'poisson flats in euclidean spaces'," *Stochastic Geometry*, pp. 202–227, 1974.
- [24] M. Otte and E. Frazzoli, "RRT X: Real-time motion planning/replanning for environments with unpredictable obstacles," in *Eleventh Workshop on the Algorithmic Foundations of Robotics*, 2014.
- [25] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, May 2012, pp. 2537–2542.
- [26] V. T. Rajan, "Optimality of the Delaunay triangulation in \mathbb{R}^d ," *Discrete and Computational Geometry*, vol. 12, no. 1, pp. 189–202, Dec. 1994.
- [27] E. Rimón and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, Oct. 1992.

- [28] J. A. Sethian, "Fast marching methods," *SIAM Review*, vol. 41, no. 2, pp. 199–235, 1999.
- [29] J. A. Sethian and A. Vladimirsky, "Fast methods for the Eikonal and related Hamilton-Jacobi equations on unstructured meshes," *Proceedings of the National Academy of Sciences*, vol. 97, no. 11, pp. 5699–5703, May 2000.
- [30] J. R. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," *Computational Geometry*, vol. 22, no. 1-3, pp. 21–74, May 2002.
- [31] A. Stentz, "The focussed D* algorithm for real-time replanning," in *International Joint Conference on Artificial Intelligence*, Aug. 1995.
- [32] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, Apr. 2010.
- [33] D. S. Yershov and E. Frazzoli, "Asymptotically optimal feedback planning: FMM meets adaptive mesh refinement," in *Eleventh Workshop on the Algorithmic Foundations of Robotics*, 2014.
- [34] D. S. Yershov and S. M. LaValle, "Simplicial Dijkstra and A* algorithms: From graphs to continuous spaces," *Advanced Robotics*, vol. 26, no. 17, pp. 2065–2085, Oct. 2012.